

# Artificial Intelligence, Algorithmic Pricing, and Collusion

Emilio Calvano, Giacomo Calzolari, Vincenzo Denicolò, and Sergio Pastorello

---

Gabe Sekeres and Finn Ye

April 23, 2025

Cornell University

# Background

---

# Motivation

- Algorithms becoming increasingly prevalent in practice
  - German gasoline markets (Assad et al. 2024)
  - Smartphone price discrimination (Kehoe et al. 2020)
- Regulatory questions:
  - How do algorithms get to collusive prices?
  - Can they do so in the absence of active principals?
  - Is algorithmic collusion visibly different than tacit collusion?
- Massive lack of theoretical guarantees for this (see Banchio and Mantegazza (2023); Possnig (2024); Lamba and Zhuk (2025))

- We're familiar with Q-learners
- Specifically, they learn *slowly*. This example has a massive state space and is trying to learn the opponent's policy as part of the state
- Our biggest criticisms are related to this. Specifically:
  - What is the loss in the learning phase?
  - How sensitive are these results to the initialization of the Q-matrix?

# Model

---

# Environment

Canonical oligopoly pricing game, with  $n$  firms / products and an outside good, where in each period  $t$  the demand for good  $i$  is

$$q_{i,t} = \frac{e^{\frac{a_i - p_{i,t}}{\mu}}}{\sum_{j=1}^n e^{\frac{a_j - p_{j,t}}{\mu}} + e^{\frac{a_0}{\mu}}}$$

where  $a_i$  is an index of quality,  $\mu$  is in index of differentiation, and  $a_0$  is an outside good. Firms choose  $p_{i,t}$ , and we have exogenous marginal costs  $c_i$ . The stage problem is:

$$\max_{p_{i,t}} q_{i,t}(p_{i,t}) \cdot p_{i,t} - q_{i,t}(p_{i,t}) \cdot c_{i,t}$$

This is quasiconcave but does not in general have a nice closed form solution.

## Simplified Stage Environment

Assume  $n = 2$ ,  $c_i = 1$ ,  $a_i = 2$ ,  $a_0 = 0$ , and  $\mu = \frac{1}{4}$ . Then the stage game reduces to

$$\max_{p_i} \frac{(p_i - 1)e^{8-4p_i}}{e^{8-4p_i} + e^{8-4p_j} + 1}$$

This is strictly concave, and we have that it admits Nash prices

$$p_i^N = p_j^N \approx 1.473$$

and monopoly prices are obtained from setting  $n = 1$ , where we attain

$$p^M = \frac{5}{4} - \frac{1}{4}W_n(2e^3) \approx 1.925$$

## Simplified Stage Environment

We basically have an extension of a Prisoner's Dilemma:

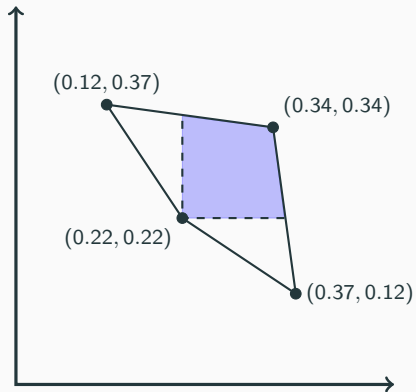
	$N$	$M$
$N$	(0.22, 0.22)	(0.37, 0.12)
$M$	(0.12, 0.37)	(0.34, 0.34)

Since all of the involved functions are continuous and concave, this extends fairly nicely.

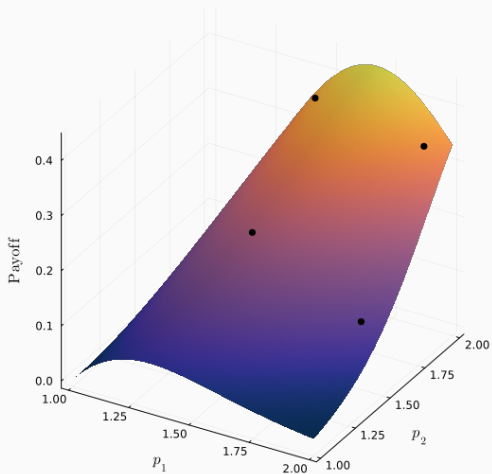
So we're making our Q-learners play a repeated Prisoner's Dilemma, and the strategies they learn *should* be similar to canonical repeated PD strategies.



# Folk Theorem



# Continuous Stage Payoffs



## A Question

Why use this sigmoid demand function instead of exogenously imposing a reasonable range for prices and using e.g. linear demand?

We don't understand what the gain from this functional form is, and the fact that it doesn't in general have closed-form solutions is an annoyance.

# Learning Theory

---

# Learning in Repeated Games

Essentially, take the opponent's previous actions to be the state, along with whatever game parameters you need. Two issues:

1. The state space is increasing as the game continues.  
Solution: Bounded memory.
2. The optimization problem is non-stationary if the opponent(s) change strategy over time.  
No official solutions here, this is why we don't have theoretical guarantees<sup>1</sup>

---

<sup>1</sup>I'm fairly sure there should be something here. At least in probability. I'm confused why nobody has proved that yet - Gabe

# Learning in Repeated Games

The Q-learners solve

$$Q(s, a) = \mathbb{E}(\pi \mid s, a) + \delta \mathbb{E} \left[ \max_{a' \in A} Q(s', a') \mid s, a \right]$$

where  $a \in A$  is the action (from the rules of the game),  $s \in S$  is the state (defined as all player actions in the last  $k$  periods, where  $k$  is the memory). Once we discretize,  $Q \in \mathbb{R}^{|A| \times |S|} = \mathbb{R}^{|A| \times |A|^{nk}}$

For simplicity,  $k = 1$ . Results robust to higher  $k$ .

## Parameterization

Work in the simplified game as above, with  $\delta = 0.95$  and  $|A| = m = 15$ . Discretize the price grid over

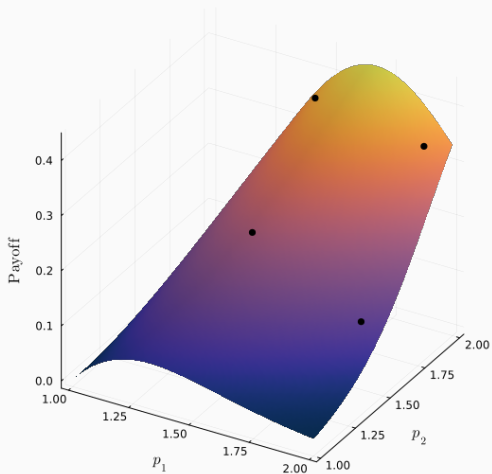
$$\left[ p^N - 0.1(p^M - p^N), p^M + 0.1(p^M - p^N) \right]$$

Set the initial matrix to the discounted payoff if the other player randomized over all actions:

$$Q_{i,0}(s, a_i) = \frac{\sum_{a_{-i} \in A^{n-1}} \pi_i(a_i, a_{-i})}{(1 - \delta)|A|^{n-1}}$$

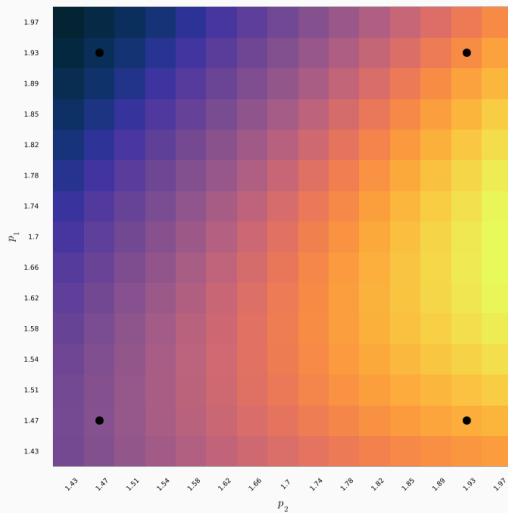
Draw the initial state  $s_0$  randomly as well.

## Continuous Stage Payoffs





# Discretized Stage Payoffs



# Parameterization

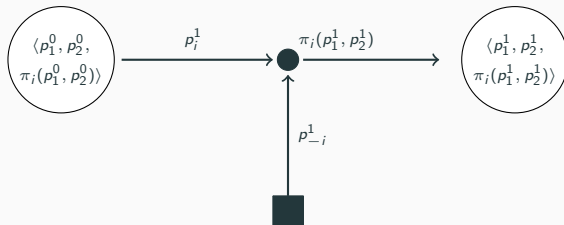
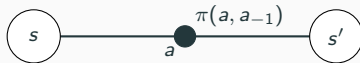
We will use  $\varepsilon$ -greedy learners, with  $\varepsilon_t = \exp(-\beta t)$ . The learning parameter  $\alpha$  will be tested over a grid of 100 points in  $[0.025, 0.25]$ , and several different values of  $\beta$  are also tested.

They define  $\nu$  to be the number of times a certain cell is visited in expectation under a certain  $\beta$ . For our purposes,  $\beta$  will be tested over a grid of 100 points in  $(0, 2 \cdot 10^{-5})$ .

Recall from earlier:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[ \pi(s', a') + \delta \max_{a' \in A} Q(s', a') - Q(s, a) \right]$$

# Learning Visualization



## Remark

In the paper, they say that their results are robust to many specifications of  $Q_0$ , the initial matrix.

We did not see that.

We saw results that changed a lot depending on  $Q_0$ .

Specifically, their results were *only* replicated by their specific choice of  $Q_0$ . If you initial  $Q_0 = 0$  for all values, they converge to the static Nash payoffs necessarily.

## Remark

Prior to this, we've generally dealt with RL algorithms that are trying to learn **payoffs** in a static or stochastic game. These algorithms are (theoretically) learning **strategies** for the infinitely repeated game.

Thinking about the bounded memory, that's no longer such an innocuous assumption. For example: these algorithms will be able to learn tit-for-tat or Grim Trigger, but not trigger strategies with  $n$  periods of punishment for  $n > k$ .

## Code

---

# Results

---